Remote Multiple Access of CD-ROM Information over a Network of DOS PCs

A thesis submitted
in partial fulfillment
of the requirements
for the degree of

Master of Technology

by

Navdeep Sood

to the

Department of Computer Science and Engineering

Indian Institute of Technology, Kanpur

March, 1994

CSE-1994-M-500-REM

23.3.94
[illegible]

CERTIFICATE

It is certified that the work contained in the thesis entitled **Remote Multiple Access of CD-ROM Information over a Network of DOS PCs**, by **Navdeep Sood** has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

Rajat Moona,
Assistant Professor,
Department of Computer Science
and Engineering,
IIT Kanpur

K R Srivathsan

K. R. Srivathsan,
Professor,
Department of Electrical
Engineering,
IIT Kanpur

March, 1994

Abstract

This project aims at designing and developing software for remote multiple access of CD-ROM information over a network of DOS Personal Computers. With its help several users can refer to refer to a remote CD-ROM database in the same session. A request-reply based client server approach was used. The client and the server were implemented over the IP layer of PCIP. The client software resides on the user PC and functions as a handler for CD-ROM system calls made by the CD-ROM based application being run by the user. All requests for CD-ROM data are redirected in a transparent manner over the network to the server software running on the server machine. The server machine has the CD-ROM mounted on a local CD-ROM drive. It executes the CD-ROM system call from the client machine on the CD-ROM under its control and reports the results back to the client machine. The client software on the client machine, which had been been waiting for the reply after having despatched the request, extracts the data in the reply packet and hands it over to the application program. The application program then proceeds with its execution as if its CD-ROM system call had been serviced locally. A stateless protocol allows easy recovery from client and server crashes. Such software would be a useful tool in setting up CD-ROM based Electronic Library services.

ACKNOWLEDGEMENT

I am grateful to my thesis supervisors Dr K R Srivathsan and Dr Rajat Moona for their invaluable guidance and constant encouragement throughout the course of this thesis. I would like to thank the ERNET and CSE lab staff for being very cooperative.

I thank Dr T V Prabhakar, Dr Gautam Barua and Mr Amitabha Roy for providing me with immediate answers to my questions in spite of their busy schedules.

Navin Kumar Saxena, Deepankar Bairagi, Vivek Magotra, Shreesh Jadhav, Deepak Gupta and Tarun Anand were always around to help me.

NAVDEEP SOOD

March, 199[illegible]

Contents

List of Figures

Chapter 1

Introduction

Today a PC on the network provides on-line access to remote databases. CD-ROM technology has developed parallelly as a way of storing huge amounts of data and a wide variety of library information is increasingly becoming available on this medium[Fox88]. Allowing a CD-ROM database to be accessed over a computer network would make it possible to share and distribute huge amounts of information inexpensively. Providing such network access would be an effective tool in implementing Electronic Library services.

Presently, remote reading of information is possible from file-servers, relational database management systems and some Internet utilities like Gopher[McC92] running under UNIX like operating systems.

In contrast to multitasking operating systems like UNIX, DOS is a non-multitasking operating system. In DOS there is no in-built facility for remote reading of information stored in disks or other devices. The sharing of data has to be accomplished through file servers running on top of DOS. For the lack of a standard for file sharing among machines running under DOS, each of these programs follows its own, usually proprietary, protocol for communication between the various components of the distributed system.

In this thesis an attempt has been made to provide an efficient yet simple utility for remote access to data stored on CD-ROMs in DOS machines. Such a system which provides remote simultaneous access to several users would be a highly economical and convenient way of sharing data and also be an effective tool in building an Electronic Library.

1.1 An Overview of Information Sharing Methods

As computer networks keep growing to reach thousands of sites and million of users around the world, it is becoming increasingly important as well as difficult to share information between computer users. Several approaches that have been employed are discussed in this section.

1.1.1 Through Distributed File Systems

Distributed file systems are important mechanisms for sharing data in an environment. They allow applications programs to access remote files as if they were stored locally. Sun NFS[San85] and Andrew File System [Mor86] are two such systems. NFS client and server for sharing disk files in DOS environment have been implemented by Bhaskar and Pawan [Bha92], and Saxena[Sax93] respectively. A file server for DOS running on UNIX was designed and implemented by Maniyar[Man91].

Any file sharing system that allows a PC file-system to be shared will make available the CD-ROM data on the network since DOS makes a CD-ROM appear as a read-only drive. The most widely sold and used such software is Novell Netware.

Netware is an operating system that allows a set of PCs to be linked together for sharing files and peripherals between then. It also supports multitasking.

It runs in two modes, dedicated and non-dedicated. In dedicated mode, Netware is the sole operating system in charge of the machine while in non-dedicated mode, it is loaded on top of DOS. This allows other DOS application programs to run concurrently with the Netware server. More details can be found in [Cro91].

Due to licensing fee etc., file servers like Netware are expensive. Our goal is to provide a low cost CD-ROM sharing software.

1.1.2 Through a Relational Database Management System(RDBMS)

The information is stored on a remote machine with sufficient hard disk capacity. The user establishes a connection with the machine, say by a remote-login, and queries the database using any of the several querying modes supported by the package. RDBMS packages are designed to manage highly structured information that can be easily visualized as a set of tables. A column of the table can have entries belonging to only one of the limited number of data types supported by the particular package. For example, ORACLE version 6.0 provides for 14 data-types like "Alpha", "Char", "Date", "Number", etc. Handling non-ASCII information is difficult, if not impossible, using such systems. Besides, these packages are expensive in terms of the hard-disk resources. They provide for easy and efficient selective retrieval of information by indexing of data and optimization of queries; by implementing a high level language for specification of queries; and by having forms based user interaction. Commercial RDBMSs have their greatest application in business related data processing.

A multiple user RDBMS needs considerable computing resource. PC based RDBMSs are considerably less sophisticated than the standard ones.

1.1.3 Distributed Information Services like Gopher and WAIS

Gopher[McC92] and WAIS[Kah91] are ways of accessing remote data in a WAN environment. Several such services have been listed and compared in [Sch92]. Data sharing in such systems is accomplished by explicit user action. They allow file transfer and execution of some pre-specified queries at a remote machine. They are limited to UNIX platforms on the Internet.

A protocol called RDAEM (Remote Database Access using Electronic Mail) has been proposed in [Jha94] which specifies the format of messages exchanged between the requesting machine and the responding machine. Using Email for remote database access is an effective way of handling services that do not demand immediate response or when there is no dedicated 24-hour communication link between the interacting systems.

1.2 Motivation

With remote multiple access possible, an extensive, economical and easy to use CD-ROM database reference service can be set up. The heart of such a system would be an array of CD-ROM servers, each capable of supporting several remote browsing sessions of any of the CD-ROM disks under its control. Each CD-ROM disk would reside with some server. The user would only have to name the database of interest to be served by the corresponding server. The server machine could be just a PC-motherboard running the server software all the time. This bank of dedicated servers providing the desired CD-ROM data on demand can be compared to the juke box of olden days which provided its user any record from those in its pack on demand and for a small fee. Figure 1.1 shows a typical CD-ROM based electronic library scenario. Two server machines are supporting three clients. Client PC1 is accessing information from two different servers in two different screen windows. The CD-ROM on Server 1 is being read by all the three Clients at the same time. A dashed line between two machines indicates flow of information between them.

1.3 Organization of the Thesis

In Chapter 2, we look at the essentials of the way DOS handles a CD-ROM and we discuss the various approaches to the design of a distributed system like ours and propose a design. Chapter 3 provides implementational details about the chosen design. A description of the software that constitutes the remote multiple access system follows in Chapter 4. In Chapter 5 we conclude with suggestions for further work.

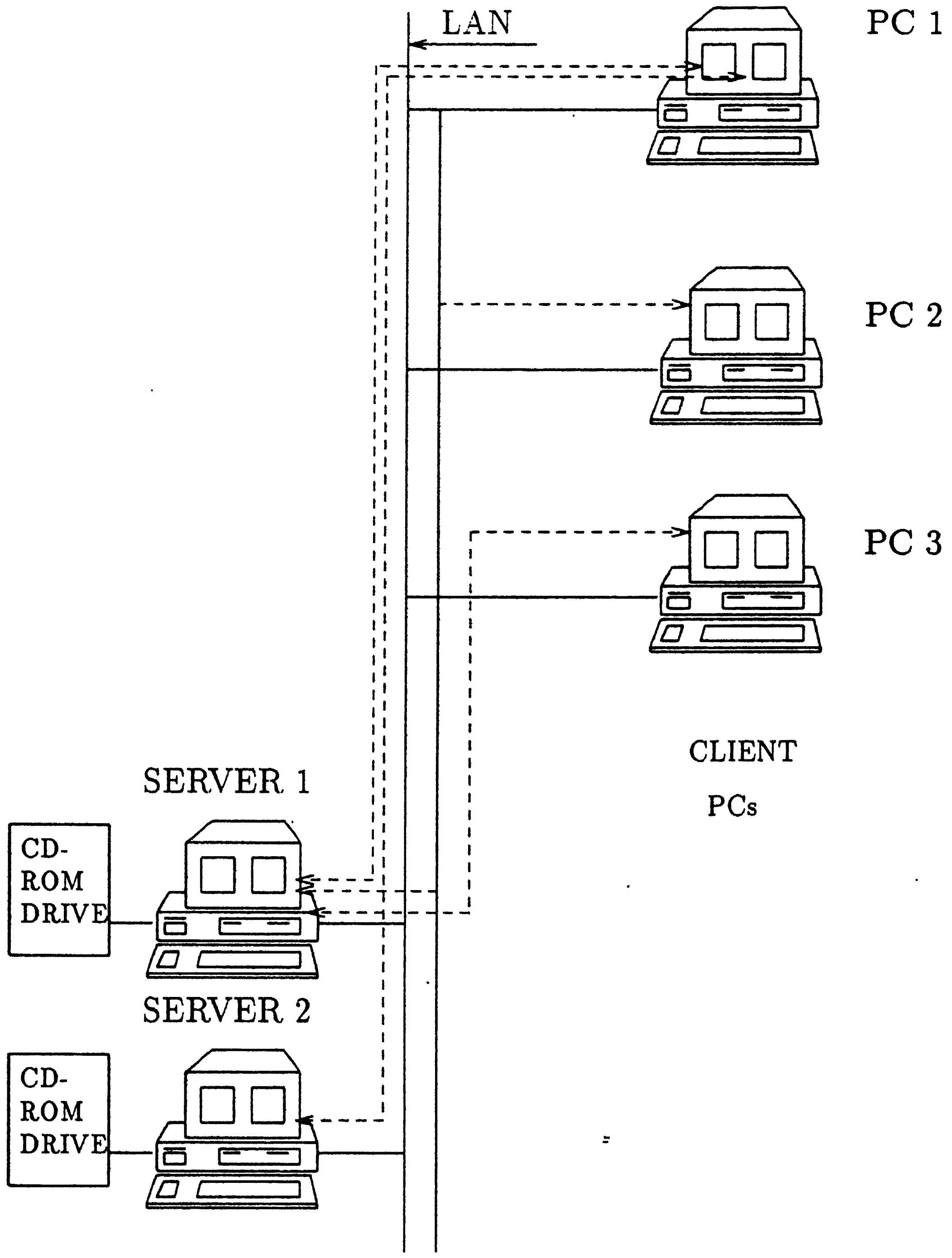


Figure 1.1: A CD-ROM Based Electronic Library

Chapter 2

Design Considerations

2.1 CD-ROM under DOS

DOS supports a CD-ROM drive using a program supplied by Microsoft and a hardware-dependent device driver supplied by the drive manufacturer. The program supplied by Microsoft is named MSCDEX.EXE. The entire CD-ROM (potentially all 660 megabytes) will appear to applications as a single MS-DOS drive letter. The Microsoft MS-DOS CD-ROM Extensions provide a high degree of compatibility with applications that depend on MS-DOS standard interfaces. Software developers need not do anything special for accessing CD-ROM disks; they issue the same MS-DOS OPEN and READ calls for opening any magnetic disks. Programmers can develop CD-ROM applications using standard MS-DOS tools. They need to be aware that they cannot create any temporary files or write any files in either that directory or on the entire CD-ROM disk.

The program MSCDEX.EXE is an installable file system driver implemented as a terminate and stay resident module. It is usually loaded using AUTOEXEC.BAT when the computer is booted. The hardware-dependent device driver implements basic functions to read the CD-ROM disk and is loaded with the MS-DOS CONFIG.SYS file. The Microsoft MS-DOS CD-ROM Extensions implement both the High Sierra file format and the ISO-9660 version of that standard. Further details on how MS-DOS handles the CD-ROM and a sample CD-ROM device driver can be found in Microsoft's CD-ROM Device Driver Development Kit which is sold by Microsoft.

The package for accessing CD-ROM information is specified by the disk manufacturer and is provided as a binary file or files with the disks. There is no standard for writing CD-ROM accessing software, hence the packages are highly incompatible with each other. Each is limited in application to its own CD-ROM database. Thus to reference a CD-ROM database the user should be careful to use the accessing software meant for that particular database. This also means that one collects an assortment of accessing packages as wide as the variety of databases available for referencing.

2.2 CD-ROM System Calls

CD-ROM related system calls are made by trapping to interrupt 2fh. Interrupt 2fh is the multiplex interrupt and is becoming more commonly used as interrupt 21 functions are getting to be in short supply. As there are several handlers that are linked to this interrupt, the higher byte of register AX (called AH hereafter) identifies which handler is to handle the interrupt. The value in AH is called the multiplex number. The specific function to be performed by the handler is placed in the lower byte of register AX. The Programmer's Technical Reference[Wil85] details the CD-ROM services provided by MS-DOS CD-ROM Extensions through interrupt 2fh. This information is given in Appendix A.

2.3 Approaches to Distributed System Design

Implementing a remote service over a computer network is a problem typical to Distributed Systems. It follows from the fact that we are using loosely coupled machines that are fundamentally independent to interact to a limited degree when there is a necessity. Depending upon the nature of the communication protocol between the cooperating processes several approaches to the design of the software are possible. A good discussion on the various aspects of Distributed Systems can be found in [Tan92]. The following approaches are commonly adopted.

2.3.1 Layered Protocol

The protocol employed has levels or layers conforming to the Open Systems Interconnection Refence model as specified in ISO 7498[ISO84] and commonly abbreviated to the OSI model. Accordingly it is divided into seven layers with each layer looking after some aspect of communication. The main problem to be overcome is to transport the bits reliably over poor physical lines. The layering could also follow some other model like TCP/IP. Relatively slow wide-area distributed systems use layered protocols extensively.

2.3.2 Remote Procedure Calls

Remote Procedure Calls (RPC) is a more subtle way of providing communication. The programs are allowed to call procedures located on other machines. Information is passed from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the programmer. The price paid for this sophistication is a high overhead of RPC implementation. Locating a server, passing complex data objects, use of global variables and handling machine crashes are some tricky problems in the implementation. Still, many distributed applications like Sun NFS are RPC based.

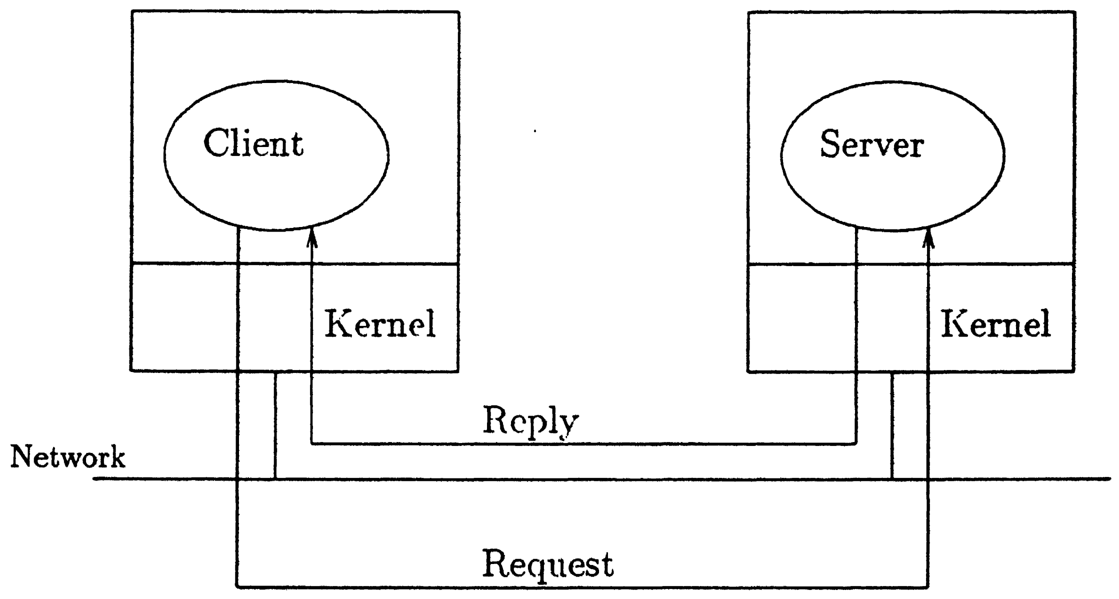
2.3.3 Simple Client Server Protocol

The distributed system is structured as a set of cooperating processes called servers that offer services to the users called clients. The client and server machines normally run under the same operating system. In a general client-server set up a machine can run a single client or server process or any mixture of the two. The communication between the cooperating processes is usually based on a connectionless request/reply protocol. Often the full OSI model is not adhered to and only a small subset of the layers is employed. See Figure 2.1 for a representaion of this type of communication. This avoids the substantial overhead associated with layered protocols for LAN based distributed systems since the physical data transfer over a LAN is far more reliable than over a WAN. By conceptually presenting interprocess communication to the programmer, the CPU processing related to the Remote Procedure Calls mechanism is also avoided. Thus for a LAN based system the request/reply based client-server model is efficient.

2.3.4 Assumptions and Features of the Design Chosen

It has been assumed that CD-ROMs are mounted on attended remote PCs operating in a DOS environment. Further that these and the client PCs are connected by a TCP/IP network. The minimal TCP/IP support needed on both the CD-ROM PC and the user PC is a connectionless service or UDP over IP.

It has been proposed that each of the PC's, physically connected to the CD-ROM drives, run under the control of a server program which would listen on the LAN for CD-ROM data requests and serve them when they arrive. The client process could run on any PC on the campus network and interface between the CD-ROM database accessing software running on the local PC and the server machine for that database in the Central Library. A server program could entertain requests from several clients pertaining to the CD-ROM drives under its control, thereby enabling several users to access the same CD-ROM database simultaneously. This would ensure high utilization of the CD-ROM databases and also save users from long waits that would be encountered in a service that entertains one user at a time. However, multiple simultaneous access will result in deterioration in the speed of service. This should be kept within acceptable limits by exercising control over the maximum number of clients that can be served simultaneously. A stateless request/reply protocol is employed between the client and the server processes so that recovery from machine crashes is uncomplicated. The following chapter gives an outline of the implementation and working of this approach.



The Client Server Model

Figure 2.1: A Client Server Pair

Chapter 3

Implementation

The Client and the Server programs have been implemented on top of the Internet layer of PCIP. PCIP is the implementation of ARPANET's TCP/IP protocol on PC in DOS environment[Rom86]. The network part of PCIP includes the Internet layer, UDP layer and the Net Interface. The Net Interface layer is responsible for packet reception and retransmission through the Net Interface card. The Internet layer does the required routing, translation etc., whenever a packet is received or has to be despatched. Upon packet reception it is passed to a user specified packet handler. The UDP layer is the topmost layer of PCIP. As with the Internet layer, a user specified handler function in invoked upon the reception of a UDP packet. All the application programs are built as specified handler functions on top of the Internet layer or the UDP layer.

PCIP does not support fragmentation and reassembly at the IP layer. This limits the maximum IP packet size or UDP packet size , including headers, to the maximum ethernet packet size of 1536 bytes. This limitation puts a rather low upper limit on the amount of data that a PCIP packet can carry. In our design, the maximum data that any packet carries has been kept to 1042 bytes.

The overall design and implementation has three major pieces. The Protocol, the Server and the Client. A summary of the operation of the system in the form of the sequence of steps that make up one remote access cycle is depicted in Figure 3.1.

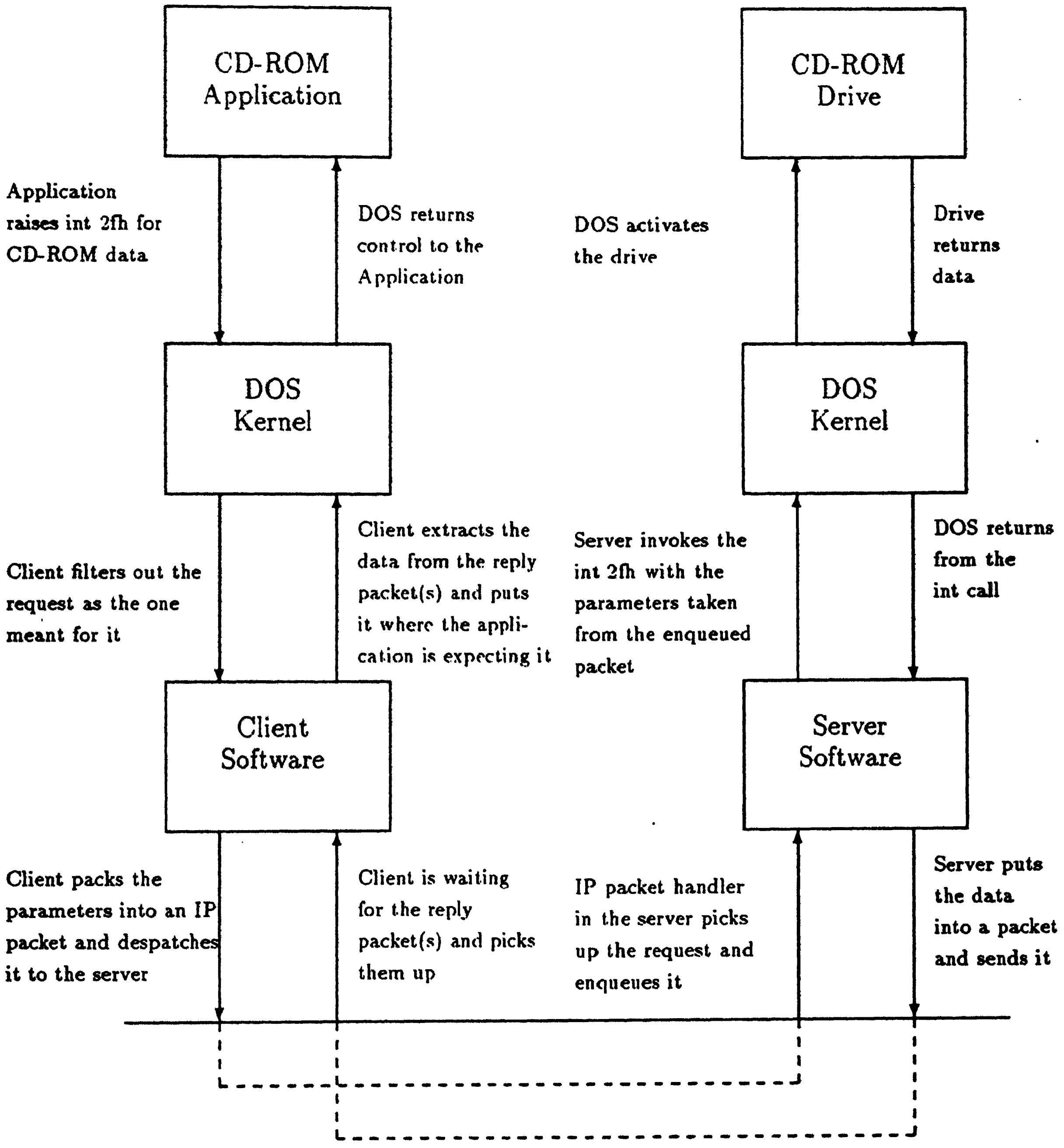

CD-ROM Application
CD-ROM Drive
Application raises int 2fh for CD-ROM data
DOS returns control to the Application
DOS activates the drive
Drive returns data
DOS Kernel
DOS Kernel
Client filters out the request as the one meant for it
Client extracts the data from the reply packet(s) and puts it where the application is expecting it
Server invokes the int 2fh with the parameters taken from the enqueued packet
DOS returns from the int call
Client Software
Server Software
Client packs the parameters into an IP packet and despatches it to the server
Client is waiting for the reply packet(s) and picks them up
IP packet handler in the server picks up the request and enqueues it
Server puts the data into a packet and sends it

Figure 3.1: Block Diagram of the System

3.1 The Protocol

A very simple request and reply protocol is used. It is based on a connectionless packet exchange. Because of constraints imposed by PCIP, no packet carries more than 1042 bytes of total user data.

Whenever the client detects an interrupt 2fh call that pertains to CD-ROM service, it floats a request packet addressed to the server machine containing details of the service needed . The first 16 bytes in a request packet contain the values passed by the application program in the CPU registers on the client machine to DOS at the time of invoking interrupt 2fh for a CD-ROM service in the eight CPU registers namely, AX, BX, CX, DX, DI, SI, ES, and DS . A request packet may also contain any other information, such as the pathname data which is a part of the interrupt request, after the first 16 bytes. Before despatching the request packet the client software inspects the nature of the request to ascertain the number of reply packets to wait for. After despatching the request packet the client goes into a wait loop.

Each request packet to the server machine contains a request from some client. After despatching this packet the client starts waiting for the server to respond with one or two packets depending upon the type of request. A reply packet contains the CPU state that the interrupt for CD-ROM service has returned. This information occupies the first 18 bytes. The extra two bytes are for the flag register. If the request has generated more than 1024 bytes of data over the results returned in the CPU registers then a second packet is employed. No CD-ROM service request generates more than two reply packetfuls of data. The first 18 bytes of the second packet are identical to the first 18 bytes of the first packet.

3.2 The Client

The client provides a transparent interface to the remote CD-ROM data. It resides as a Terminate and Stay Resident program while the program that needs data from the remotely mounted CD-ROM runs in the foreground. The client catches all interrupt 2fh calls and for each call determines whether it is a CD-ROM related call or otherwise. If a call is found to refer to a CD-ROM service, that is, if the higher byte of register AX contains the value 15h, the client takes necessary action to request the required service from the CD-ROM server. The client blocks until the server returns the results. After receiving the reply from the server, the client hands over the results to the calling program and returns the control to it as shown in Figure 3.1. If the interrupt 2fh call is determined to be a non CD-ROM call it is chained back to the original interrupt handler.

Each invocation of the interrupt 2fh that is made with 15h value in AH generates a request-for-service packet by the client software. This packet contains the CPU state that the server would require to provide the requested information. If the server fails to send a reply within a predecided time, the request packet is retransmitted. After a finite number of unsuccessful retransmissions the client reports failure to the calling program.

Any CD-ROM request that requires long service time is broken up into a sequence of shorter remote requests by the client software. This is done in a way that is transparent to the calling program. This ensures a more equitable distribution of server time between competing clients and prevents any particular client from monopolising the server for long durations.

3.3 The Server

The server software runs continuously on the server machine as a normal DOS program. Request packets are queued as they arrive by the IP packet handler in the server as in Figure 3.1. The packet at the head of the queue is taken up for processing as soon as the previous packet is processed. Each request-for-service packet contains the CPU state of the client at the time the interrupt 2fh was invoked. Hence each packet in the queue represents an interrupt 2fh call on some client on the network, that had 15h in AH. This way several clients are able to seek service from the same server at the expense of a small and unnoticeable wait. Most of the time the wait would be insignificant, as the number of clients that would be reading data from the same CD-ROM in the same session would be only one or two. The maximum possible wait is a function of the maximum queue length allowed which can be set at compile time. Once the queue is full any further request packets are ignored.

After dequeuing a request packet the server fills up the client CPU state into an appropriate data structure and executes an "Int86" function call. This function is a part of the Run-Time library that the Microsoft C Compiler provides. Depending upon the kind of service sought one or two reply packets are despatched to the client machine. The client machine knows how many reply packets to expect from the server and waits for them. There is no remote service call that needs more than two reply packets.

The server does not wait for any acknowledgements from the clients. A client that has missed the reply packet(s) meant for it times out from the wait and resends its request.

The maximum length request packet queue in the server should not be raised to a size that makes the maximum wait time in the queue equal to or greater than the timeout duration on the client end. Otherwise there is a high possibility of a client timing out before its request packet is taken out of the queue by the server and then resending the same request. However, such an occurrence would not cause anything worse than some unnecessary work for the server.

Chapter 4

Software Description

This chapter provides a brief description of the software written to implement the design discussed in the previous chapters. The whole code forms two programs: the Client and the Server. Most of the code is in C and the remaining in assembly. Microsoft C Compiler 5.0 and Microsoft Macro Assembler 5.1 and their associated utilities were used to compile and link the two programs. Short descriptions of the procedures that make up the Client and the Server code follow.

The data structure central to the whole software is a structure that stores all the information about the particular interrupt service request being attended to. It has been typedefined in the header file cdr.h as type *req*, which stands for "request for service".

```
struct req {
    union REGS     regs;      /* for general registers */
    struct SREGS   sregs;     /* for segment registers */
    int            status;    /* request sent, reply received, timed out */
    in_name        svr_name;  /* used by client for server name */
    PACKET         req_pkt;   /* IP packet  that contains the request */
    int            req_len;   /* length of the data in the request packet */
    PACKET         rep_pkt;   /* IP packet if received in reply to request */
    int            rep_len;   /* length of the data in the reply packet */
    in_name        source;    /* used in server only to record the source of the
                                 request for transmitting back the reply to */
};
```

```
typedef struct req req;
```

The first two fields store the client CPU state at the time the CD-ROM system call was made. Their types have been defined in the header file "dos.h" that comes with the C compiler used. The next field indicates the status of the request trapped by the client: whether a request packet has been despatched to the server, whether the reply is awaited, or whether the wait for the reply has been timed out, etc. The full list of the values this field takes is in the file cln.h. The remaining fields are well described by the comments beside them.

4.1 The Client Program

As stated in chapter 3, the Client has been implemented as a terminate and stay resident program on the user machine. It traps all CD-ROM related interrupts and procures the required service from the server machine on the network. The client software is made up of several routines. This section discusses them in brief.

4.1.1 Main Function

The main procedure starts by saving the stack pointer and stack segment values and ends by making itself memory resident. In between the network communication initialization and the installation of the interrupt servicing routine are carried out.

4.1.2 nwinit()

This is the routine called by the main client procedure to initialize the network. It is called only once during the installation of the client program on the client machine with the number of PCIP packet buffers and the packet buffer size as its inputs. It contains calls to the required PCIP procedures.

4.1.3 inst_cln()

This routine is called after the network has been initialized. It makes up the second half of intialization and exits after installing the client program as an interrupt service routine. An IP connection to the server is opened and the interrupt 2fh vector is appropriated. The old value of the interrupt handler for 2fh is saved.

4.1.4 new2fh()

It contains the interrupt 2fh handler that performs the function of filtering out CD-ROM related requests and processing them. Non CD-ROM calls are passed to the previous handler whose address had been saved.

4.1.5 CD-ROM functions

There is a function for each possible CD-ROM system call (see Appendix A for a list of CD-ROM calls). When invoked, each of them sends a request packet and processes the reply packet (if it comes) before exiting. These functions are ichk(), adr(), adw(), dc(), gafn(), gbdfn(), gcddl(), gcfn(), tdoff(), tdon(), gddl(), gde(), gmv(), gsvdp(), rbms() and rvtoc(). Exactly which of these sixteen is called depends on the value of register AX (to be specific on the value of AL the lower byte of the register AX) at the time of raising of interrupt 2fh on the client machine.

4.1.6 nodata(), smalldata() and longdata()

These functions do the actual sending and receiving of the packets for the functions in the previous section. All CD-ROM system calls have been implemented by using one of the three depending upon whether the call requires no data other than the server CPU state from the server (where nodata() is used), or whether the call requires a small amount of data from the server in addition to the server CPU state from the server (where smalldata() is used) or whether a larger amount of data with the server CPU state is required (where longdata() is used). Up to 1024 bytes of data is considered "small" and up to 2048 bytes is considered "long".

4.1.7 rep_wait()

This routine is called by any procedure that is waiting for an IP packet from the server. The body of this procedure is a busy-wait loop which terminates when a certain number of iterations are over (which is the timeout situation) or when a packet is received from the server. PCIP does not permit an interrupt service routine to use PCIP timers [Rom86]. Hence this procedure uses busy waiting which instead of using a PCIP timer and yielding the processor is fine since the client machine has nothing to do while the wait for the reply is going on.

4.1.8 ip_hndlr()

This function is invoked by PCIP whenever an IP packet, which belongs to the protocol the client and server programs are using to communicate, arrives from the server. It checks if the Client program is expecting a server reply, that is if rep_wait() is executing, before accepting the packet for further processing. Otherwise the IP packet is discarded. This checking prevents snags due to delayed packets. A look at the status field of the *req* structure tells if any request for service is awaiting the server reply.

4.1.9 Stack Management Functions

The client program when called by an invocation of interrupt 2fh, switches over to its own stack from the stack of the program which has raised the interrupt. After its job is done the stack of the calling program is restored. This is done by the functions set_tsrstk() and rst_tsrstk(). sv_stk() stores the stack pointer and stack segment values during the initial phase for subsequent use.

4.1.10 Data Movement Functions

mdata(), mpname() and cp_regs() are used for moving data from one place to another. Their names are quick and unimaginative abbreviations for "move data", "move path name" and "copy registers".

4.2 Server Software

The Server is basically the interplay between the packet handler which enqueues the request-for-service packets as they arrive from various clients and the remaining routines which dequeue these packets one by one, invoke the corresponding interrupt locally, and despatch the generated results to the respective client machine.

4.2.1 Main Function

The main function intializes the network communication, installs the packet handler and starts waiting for the request packets to be put in the service queue. As soon as a waiting request is detected it is processed and the queue is readjusted. If another request arrives in the meantime it is taken up otherwise waiting commences.

4.2.2 ip_hndlr()

This function is invoked by PCIP whenever a packet on the server's IP connection is received. It enqueues the received packet and exits.

4.2.3 service()

This function is called by the main server function to process any request at the head of the queue. It in turn calls CD-ROM system call functions which invoke the specified interrupt and send back the reply.

4.2.4 CD-ROM system calls functions

These functions correspond to the client functions in number and name. That is, they are sixteen in number and are named as ichk(), adr(), adw(), dc(), gafn(), gbdfn(), gcddl(), gcfn(), tdoff(), tdon(), gddl(), gde(), gmv(), gsvdp(), rbms() and rvtoc(). Each invokes the 2fh interrupt on the server machine and despatches the results using the functions described in the next section.

4.2.5 sendnodata(), sendsmalldata() and sendlongdata()

These functions do the actual invocation of the interrupt and the subsequent processing of the data got from the CD-ROM. One of them is used by each CD-ROM system call function of the above section depending on the amount of the data, in addition of the server CPU state, that has to be sent to the client. sendnodata() sends only the CPU state, sendsmalldata() is for additional data till 1024 bytes over it and sendlongdata() is for up to 2048 bytes of data over the basic CPU state.

4.2.6 Other functions

These are small functions that do miscellaneous jobs. They include the data movement functions and a function to despatch an IP packet. The data movement functions are mdata(), mpname() and cp_regs(). They are identical to their client counter parts save for cp_regs which is slightly different. despatch() is used to despatch IP packets.

4.3 Compiling and Running the Software

The Client code resides in the directory "cli". To compile the Client use the makefile clnmake. The Client code is compiled and linked into cln.exe. Boot up the client machine using the same start up files AUTOEXEC.BAT and CONFIG.SYS as if the CD-ROM drive were local to the client. Install the Client on the client PC by executing cln from the DOS prompt.

The Server code is in the directory "server". It can be compiled by using srvmake. The executable file is called srv.exe.

Read the README files in the directories before compiling the programs.

Chapter 5

Conclusions

Software for remote multiple access of CD-ROM information over a network of DOS Personal Computers was designed and coded. It has a Client part which runs on the user PC and a Server part that runs on the server PC. They communicate over the network using a a request/reply protocol. The Client and the Server were implemented over the IP layer of PCIP to have a faster response than a UDP based implementation.

The Client software installs itself as a Terminate and Stay Resident program on the client PC and it functions as an interrupt handler for interrupt 2fh through which CD-ROM system calls are made. All requests for CD-ROM data are identified and redirected by the Client in a transparent manner over the network to the Server software running on the server machine. The Server machine has the CD-ROM mounted on a local CD-ROM drive. The Server software extracts the parameters of the call from the request packet received from the client machine and executes the call on the CD-ROM under its control and reports the results back to the Client . After receiving the reply from the server, the Client extracts the data in the reply packet and hands it over to the application program. The application program then proceeds with its execution as if its CD-ROM system call had been serviced locally. A stateless protocol permits uncomplicated recovery from client and server crashes.

Because of non-availability of a PC with a CD-ROM drive on the network the design was tested in a limited way using two PCs which were running the Server and the Client programs respectively. The Client software became properly memory resident and filtered out all arificially raised CD-ROM system calls. Futher testing, on the full hardware configuration assumed in the design, is needed. When being tested fully in future, the software should be modified, if necessary, to reflect the changes in the specifications of the CD-ROM system calls from the ones used in this project. See Appendix A for a complete description of the CD-ROM system calls implemented.

Further extentions and additions to the work in this thesis that would make the software developed a better Electronic Library utility are:

(1) Integrating the Client code with a widowing environment to allow the user to run more than one CD-ROM application simultaneously and independently of each other.

(2) The Server program can have CD-ROM usage statistic collection added to it. It would help in the management of the CD-ROM based services by providing data which can be analyzed to ascertain user preferences and other such information.

(3) User authentication measures can be added so it becomes possible to provide selective access.

The benefits of remote multiple access to CD-ROM information in buliding an Electronic Library system have been discussed earlier. This work can be a starting platform for setting up a shared bank of CD-ROM databases and be the first step towards an Electronic Library.

Bibliography

[Bha92] Bhaskar, H., Pawan, G., *NFS Client for PC in DOS environment*, B. Tech Thesis, IIT Kanpur, Apr 1992.

[Cro91] Croucher, P., *Novell Netware Companion*, Galgotia Publications, 1991.

[Fox88] Fox, E.A., *Optical Disks and CD-ROM: Publishing and Access*, Annual Review of Information Science and Technology, Vol 23, 1988.

[ISO84] *ISO 7498, Information Processing Systems - Open Systems Interconnection - Basic Reference Model*, 1984.

[Jha94] Jha, S.K., Prabhakar, T.V., *Querying Remote Electronic Libraries*, In preparation, March 1994.

[Kah91] Kahle, B., Medlar, A., *An Information System for Corporate Users: Wide Area Information Servers. ConneXions - The Interoperability Report*, 5(11), Interop Inc, Nov 1991.

[Man91] Maniyar, S.N., *A Remote DOS Disk Server on UNIX Machine*, M. Tech. Thesis, IIT Kanpur, 1991.

[McC92] McCahil, M., *The Internet Gopher: A Distributed Server Information System, ConneXions - The Interoperability Report*, 6(7), Interop Inc, July 1992.

[Mor86] Morris, J.H., Satyanarayan, M.,Conner, M.h., Howard, J.H., Rosenthal, D.S., Smith, F.D., *Andrew : A Distributed Personal Computing Environment*, Communications of the ACM Vol 29, No 3, March 1986.

[Rom86] Romkey, J.L., *The PC/IP Programmer's Manual*, Unpublished Documentation, Laboratory of Computer Science, M.I.T., Jan 1986.

[San85] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B., *Design and Implementation of the Sun Network Filesystem*, Summer Usenix Conference proceedings, June 1985.

[Sax93] Saxena, N.K., *An NFS Server under DOS*, M. Tech Thesis, IIT Kanpur, 1993.

[Tan92] Tanenbaum, A. S.,*Modern Operating Systems*, Prentice Hall, 1992.

[Wil90] Williams, D., *The Programmer's Technical Reference : MS-DOS , IBM PC Compatibles*, Galgotia Publications, 1990.

Appendix A

CD-ROM System Calls

Interrupt 2fh CD-ROM calls are detailed under. These specifications have been taken from [Wil90].

```
Function    15h    CD-ROM extensions
            Microsoft CD-ROM driver versions 1.0 through 2.0 will work only up
            to DOS 3.31.  DOS 4.0 and up require 2.1 drivers.

entry    AH    15h    CD-ROM services
         AL    subfunctions
               00h       Installation Check
               BX        00h
               return    BX       number of CD-ROM drive letters used
                         CX       starting drive letter (0 = A:)
               note      This installation check DOES NOT follow the format
                         used by other software.

               01h       Get Drive Device List
               ES:BX     pointer to buffer to hold drive letter list
                         (5 bytes per drive letter)
               return    buffer filled, for each drive letter :
                         byte     subunit number in driver
                         dword    address of device driver header
```

```
02h     Get Copyright File Name
CX      drive number (0 = A)
ES:BX   pointer to 38-byte buffer for name of copyright file
return  CF      set if drive is not a CD-ROM drive
        AX      error code (15h)


03h     Get Abstract File Name
CX      drive number (0 = A)
ES:BX   pointer to 38-byte buffer for name of abstract file
return  CF      set if drive is not a CD-ROM drive
        AX      error code (15h)


04h     Get Bibliographic Doc  File Name
CX      drive number (0 = A)
ES:BX   pointer to 38-byte buffer for name of bibliographic
        documentation file
return  CF      set if drive is not a CD-ROM drive
        AX      error code (15h)


05h     Read VTOC (Volume Table of Contents)
CX      drive number (0 = A)
DX      sector index (0 = first volume descriptor,
                          m1 = second, ...)
ES:BX   pointer to 2048-byte buffer
return  CF      set on error
                AX    error code (15h, 21h)
        CF      clear if successful
                AX    volume descriptor type (1 = standard,
                      0FFh = terminator, 00h = other)


06h     Turn Debugging On
```

BX	debugging function to enable		
note	Reserved for development.		

07h	Turn Debugging Off		
BX	debugging function to disable		
note	Reserved for development.		

08h	Absolute Disk Read		
CX	drive number (0 = A:)		
DX	number of sectors to read		
ES:BX	pointer to buffer		
SI:DI	starting sector number		
return	CF	set on error	
	AL	error code (15h, 21h)	

09h	Absolute Disk Write		
CX	drive number (0 = A:)		
DX	number of sectors to write		
ES:BX	pointer to buffer		
SI:DI	starting sector number		
note	Corresponds to int 26h and is currently reserved and nonfunctional.		

0Ah	Reserved by Microsoft		

0Bh	CD-ROM 2.00 - Drive Check		
CX	drive number (0 = A:)		
return	BX	0ADADh if MSCDEX.EXE installed	
	AX	0	if drive not supported
		<> 0	if supported

0Ch	CD-ROM 2.00 - Get MSCDEX.EXE Version		

return	BH	major version	
	BL	minor version	
note	MSCDEX.EXE versions prior to 1.02 return BX = 0.		

0Dh	CD-ROM 2.00 - Get CD-ROM Drive Letters		
ES:BX	pointer to buffer for drive letter list (1 byte per drive)		
return	buffer filled with drive numbers (0 = A:). Each byte corresponds to the drive in the same position for function 1501h.		

0Eh	cd-ROM 2.00 - Get/Set Volume Descriptor Preference		
BX	subfunction		
	00h	Get Preference	
	DX	00h	
	return	DX	preference settings
	01h	Set Preference	
	DH	volume descriptor preference	
		1	primary volume descriptor
		2	supplementary volume descriptor
	DL	supplementary volume descriptor preference	
		1	shift-Kanji
	CX	drive number (0 = A:)	
	return	CF	set on error
		AX	error code (01h, 15h)

0Fh	CD-ROM 2.00 - Get Directory Entry		
CX	drive number (0 = A:)		
ES:BX	pointer ASCIIZ pathname		
SI:DI	pointer to 255-byte buffer for directory entry		
return	CF	set on error	
	AX	error code	

```
        CF       clear if successful

        AX       disk format (0 = High Sierra, 1 = ISO 9660)

note    Directory entry format :

        byte     length of directory entry

        byte     length of XAR in LBN's

        dword    LBN of data, Intel (little-Endian) format

        dword    LBN of data, Motorola (big-Endian) format

        dword    length of file, Intel format

        dword    length of file, Motorola format

---High Sierra---

      6 bytes    date and time

        byte     bit flags

        byte     reserved

---ISO 9660---

      7 bytes    date and time

        byte     bit flags

---both formats---

        byte     interleave size

        byte     interleave skip factor

        word     volume set sequence number, Intel format

        word     volume set sequence number, Motorola format

        byte     length of file name

      n bytes    file name

        byte     (optional) padding if filename is odd length

      n bytes    system data


Error codes :

        01h      invalid function

        15h      invalid drive

        21h      not ready
```